Yale University

Diffusion Models

S&DS 265 · Introductory Machine Learning · Lecture 18

Zhuoran Yang

Fall 2026

AI usage: figures were generated, and these slides polished, with the help of AI tools.

Outline

Learning Objectives

In this lecture you will learn:

1. Why generation can be organized as a path between the data and pure noise, with the forward direction free of any training;
2. How Gaussian corruption gives a closed-form noising law at every level;
3. Why the derivation must condition on x_0, and why doing so is legitimate;
4. How the variational bound collapses into one line of noise regression, and what the training and sampling loops then look like in PyTorch;
5. Why the denoiser is a U-Net, and what its skip connections, attention block, and time embedding each contribute.

Outline

Text-to-Image Generation

prompt: ``*an astronaut riding a horse*''

(no style clause) ``, by da Vinci'' ``, by Hiroshige'' ``, by Picasso''

Stable Diffusion, 2022, by Wikimedia Commons users Asanagi and VulcanSphere; CC0 and public domain. Prompt and model recorded on each file page.

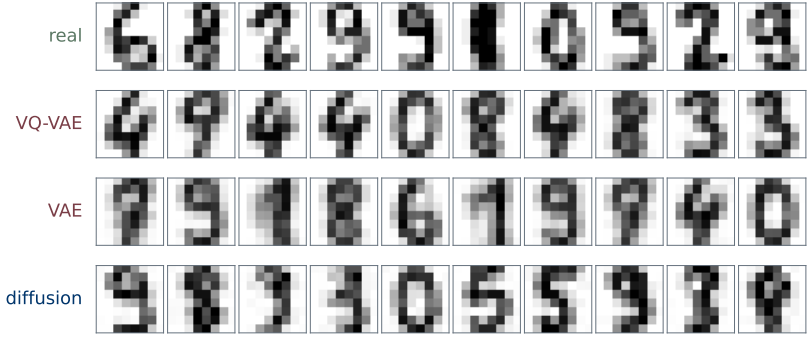
Diffusion Models in Practice

Three frames from a single generated clip. The same construction drives Stable Diffusion and FLUX for images, DALL·E 3 inside ChatGPT, and Sora, Veo, and Seedance for video.

In-class demonstration: `https://seed.bytedance.com/en/seedance`

Source: ByteDance Seed (2025), *Seedance 1.0*, arXiv:2506.09113, Figure 16; credited educational excerpt.

Generating Digits with the Models of Lecture 17



The same data, four ways of producing a picture. A VQ-VAE assigns each image one of K codes, so drawing a code uniformly returns at most $K = 64$ distinct images. A VAE has a prior to sample from and returns blurred averages. The diffusion model is trained with the same width, optimiser, and number of gradient steps as the VAE.

Energy distance to the data 0.135, 0.200, and 0.043. Numbers in `generator-comparison.csv`.

Two Lectures

Lecture 18 — the foundation

a path between data and noise; the forward process; the training objective and why its derivation conditions on the clean sample; training and sampling; the U-Net that carries them.

Lecture 19 — recent methods, and generating from text

sampling in far fewer steps; running the process on a learned code; a straighter path; conditioning on a text prompt and amplifying it; and diffusion models of text itself.

Scope of today. The opening example was conditional — an image *for a prompt*. Today builds the unconditional machinery, which is where the difficulty lies; the prompt is one extra network argument, and is Lecture 19.

The Digits Data

Every calculation today runs on data small enough to rerun in class.

- 1,797 handwritten digits, 8×8, rescaled to $[-1, 1]$ — the set used in the comparison just shown;
- 400 Olivetti faces, 64×64, where the exactly optimal denoiser can be written down in closed form and compared with a network;
- No labels are used: the object to model is the distribution of the pixel arrays themselves.

Data: scikit-learn handwritten digits and Olivetti faces; BSD-3-Clause distribution.

Reading One Image

One observation is the whole array, flattened:

$$x_0 \in [-1,1]^{64} \text{ (digits)}, \qquad x_0 \in [-1,1]^{4096} \text{ (faces)}.$$

- The object to model is a distribution over that whole space;
- The training set supplies a finite number of points in it, and nothing else.

A deployed system replaces 4,096 by 786,432 and 400 by hundreds of millions. Nothing else in the construction changes.

Problem Setup

Given examples $x^{(1)}, \ldots, x^{(n)}$ from an unknown p_{data},

$$\text{produce a new draw } \tilde{x} \sim p_{\text{data}}.$$

The difficulty is that p_{data} is never written down. All we ever have is a finite set of points drawn from it.

Problem Setup

Given examples $x^{(1)}, \ldots, x^{(n)}$ from an unknown p_{data},

$$\text{produce a new draw } \tilde{x} \sim p_{\text{data}}.$$

The difficulty is that p_{data} is never written down. All we ever have is a finite set of points drawn from it.

Lecture 17 already built a sampler for this problem. The VAE works — it draws a code from the prior and decodes it — but its samples are noisy, and getting high-fidelity images out of it is hard. The next two slides say why.

Why VAE Samples Come Out Blurry

Lecture 17's decoder places a Gaussian around its output,

$$p_\theta(x \mid z) = \mathcal{N}(g_\theta(z),\, \sigma^2 I), \qquad -\log p_\theta(x \mid z) = \tfrac{1}{2\sigma^2}\|x - g_\theta(z)\|_2^2 + \text{const},$$

so fitting the decoder is squared-error regression of x on z, and squared error is minimized by the conditional mean:

$$g_\theta(z) = \mathbb{E}[\,x \mid z\,] \;=\; \text{the average of every image compatible with } z.$$

Why VAE Samples Come Out Blurry

Lecture 17's decoder places a Gaussian around its output,

$$p_\theta(x \mid z) = \mathcal{N}(g_\theta(z),\, \sigma^2 I), \qquad -\log p_\theta(x \mid z) = \frac{1}{2\sigma^2}\|x - g_\theta(z)\|_2^2 + \text{const},$$

so fitting the decoder is squared-error regression of x on z, and squared error is minimized by the conditional mean:

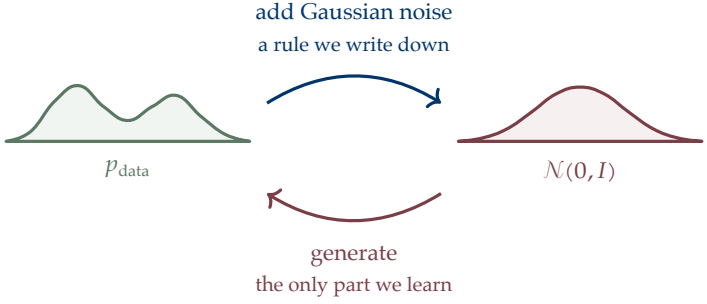
$$g_\theta(z) = \mathbb{E}[\,x \mid z\,] \;=\; \text{the average of every image compatible with } z.$$

An average of sharp images is not a sharp image. Whatever the code does not pin down gets averaged away — and the more pixels there are, the more it misses.
Diffusion models address exactly this. We return to the comparison at the end of the lecture.

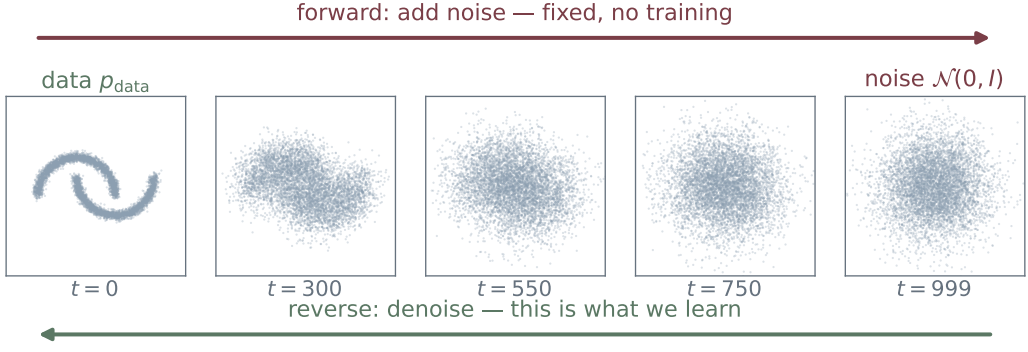
Outline

The Central Construction

Construct a path of distributions joining p_{data} to a base distribution, e.g., Gaussian, we can sample. Going from p_{data} to base distribution requires no learning; the reverse direction is the learning model.



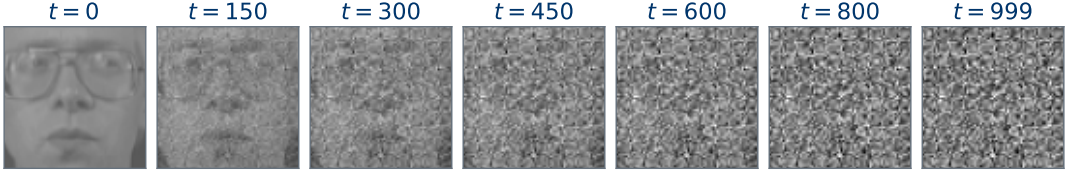
The Path of Distributions



Each panel is the distribution of the state at one time. Left to right, structure is destroyed by adding Gaussian noise — a rule we write down. Right to left is generation, and is the only part that needs a network.

Course computation: the two-moons target under a cosine schedule, seed 26521.

The Forward Path on a Real Image



One face, one fixed noise draw, seven points along the path. The label $\overline{\alpha}_t$ is the fraction of the original signal still present. By $t = 800$ a person cannot tell which face this was; by $t = 999$ neither can any statistic.

Course computation on the Olivetti faces; cosine schedule, seed 26521.

Motivation for Many Small Steps

Suppose the reverse of the whole path were learned in a single step:

$$x_T \sim \mathcal{N}(0, I) \longmapsto \tilde{x}_0.$$

That is the autoencoder decoder again, and exactly as hard.

Dividing the path into T small steps changes the question each step answers to given a slightly noisy image, remove a little noise.

Motivation for Many Small Steps

Suppose the reverse of the whole path were learned in a single step:

$$x_T \sim \mathcal{N}(0, I) \longmapsto \tilde{x}_0.$$

That is the autoencoder decoder again, and exactly as hard.

Dividing the path into T small steps changes the question each step answers to given a slightly noisy image, remove a little noise.

Each small step is a supervised learning problem. We add the noise ourselves, so we know what was added: the noisy image is the input, the noise is the label, and the fit is least squares. That is the point of imposing a forward process — not a modelling assumption about images, but a device for manufacturing labelled examples out of unlabelled data.

Outline

One Forward Step

Fix a small $\beta_t \in (0,1)$ and set $\alpha_t = 1 - \beta_t$. The forward step is

$$\boxed{x_t = \sqrt{\alpha_t} \cdot x_{t-1} + \sqrt{1-\alpha_t} \cdot \varepsilon_t, \qquad \varepsilon_t \sim \mathcal{N}(0, I) \text{ independent}}$$

or, written as a distribution,

$$q(x_t \mid x_{t-1}) = \mathcal{N}(\sqrt{\alpha_t} \cdot x_{t-1},\ \beta_t I).$$

Two things happen at once: the signal is shrunk by $\sqrt{\alpha_t}$ and fresh noise is added with standard deviation $\sqrt{\beta_t}$. The sequence $\beta_1, \ldots, \beta_T$ is ours to choose, and we return to it shortly.

Variance Preservation

derive

Suppose we only added noise, $x_t = x_{t-1} + \sqrt{\beta_t} \cdot \varepsilon_t$. Then the variance grows without bound and the state never settles anywhere we know how to sample from.

With the shrinkage, take one coordinate with $\mathrm{Var}(x_{t-1}) = 1$:

$$\begin{aligned} \mathrm{Var}(x_t) &= \alpha_t \,\mathrm{Var}(x_{t-1}) + (1 - \alpha_t)\,\mathrm{Var}(\varepsilon_t) \\ &= \alpha_t + (1 - \alpha_t) = 1. \end{aligned}$$

Variance Preservation

derive

Suppose we only added noise, $x_t = x_{t-1} + \sqrt{\beta_t} \cdot \varepsilon_t$. Then the variance grows without bound and the state never settles anywhere we know how to sample from.

With the shrinkage, take one coordinate with $\mathrm{Var}(x_{t-1}) = 1$:

$$\begin{aligned} \mathrm{Var}(x_t) &= \alpha_t \,\mathrm{Var}(x_{t-1}) + (1 - \alpha_t)\,\mathrm{Var}(\varepsilon_t) \\ &= \alpha_t + (1 - \alpha_t) = 1. \end{aligned}$$

The scale is preserved at every step: the process is **variance preserving**. The pair $(\sqrt{\alpha_t}, \sqrt{1 - \alpha_t})$ plays the role of $(\cos\vartheta, \sin\vartheta)$ — it rotates signal into noise without changing the total.

The Closed-Form Marginal

Composing t steps collapses into a single Gaussian. Writing $\overline{\alpha}_t = \prod_{s \le t} \alpha_s$,

$$\boxed{q(x_t \mid x_0) = \mathcal{N}(\sqrt{\overline{\alpha}_t} \cdot x_0,\ (1 - \overline{\alpha}_t) I)}$$

equivalently

$$x_t = \sqrt{\overline{\alpha}_t} \cdot x_0 + \sqrt{1 - \overline{\alpha}_t} \cdot \varepsilon, \qquad \varepsilon \sim \mathcal{N}(0, I).$$

So $\overline{\alpha}_t$ is the fraction of signal energy still present, and training never simulates the chain: draw t, draw one ε, and jump straight to x_t.

As $t \to T$, $\overline{\alpha}_t \to 0$ and $q(x_T \mid x_0) \to \mathcal{N}(0, I)$ for every x_0 — which is what lets generation start there.

Source: This is what makes the supervision free: every pair (x_t, ε) follows from a clean image. Proof in Appendix A.2.

Noise Scheduling

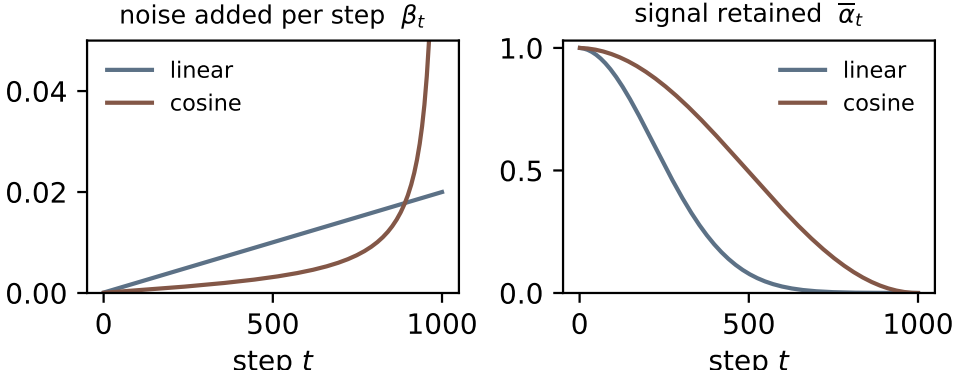
The forward process has one free ingredient: the sequence $\beta_1, \dots, \beta_T$, the noise added at each step. Everything else follows,

$$\alpha_t = 1 - \beta_t, \qquad \bar{\alpha}_t = \prod_{s \le t} \alpha_s, \qquad x_t = \sqrt{\bar{\alpha}_t} \cdot x_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \varepsilon.$$

Choosing $\{\beta_t\}$ is called the **noise schedule**. It is a design choice, not something learned, and it decides how the difficulty is spread across t.

Equivalently one may specify $\bar{\alpha}_t$ directly and read β_t back off it, which is how the cosine schedule is defined.

Linear and Cosine Noise Schedules



The linear schedule destroys the signal early: $\bar{\alpha}_t$ is near zero by $t \approx 700$, so the last 300 steps carry almost nothing. The cosine schedule spends the budget evenly. Its name comes from the signal curve on the right, which is set to a squared cosine, $\bar{\alpha}_t \propto \cos^2\left(\frac{\pi}{2} \cdot \frac{t/T+s}{1+s}\right)$.

Linear after Ho et al. (2020), cosine after Nichol and Dhariwal (2021). Both schedules are written out in Appendix A.3.

Two Noise Schedules

They are specified on different objects, which is the whole difference.

- **Linear** (Ho et al.): fix β_t itself as a straight ramp, β_t from 10^{-4} to .02 over $T = 1000$ steps.
- **Issue of linear schedule**: $\overline{\alpha}_t$ decays too fast (geometrically).
- **Cosine** (Nichol and Dhariwal): fix the signal $\overline{\alpha}_t$ instead, as a squared quarter-cosine, and calculate β_t from $\overline{\alpha}_t$.

The names follow the object each one controls: a line in β_t, a cosine in $\overline{\alpha}_t$.

Outline

The Parameterized Reverse Process

Reverse process is a learned neural network generative model p_θ.

Factorize the joint distribution in the reverse direction, and make each step a neural-Gaussian model:

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t), \qquad p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(\mu_\theta(x_t, t),\ \sigma_t^2 \cdot I).$$

We set $p(x_T) = \mathcal{N}(0, I)$. Here σ_t is fixed and we have a closed-form.

Recall that we use neural-Gaussian model in VAE. Our idea is similar here. We set $\sigma_t^2 = \sigma_q^2(t)$ in p_θ.

The Parameterized Reverse Process

Reverse process is a learned neural network generative model p_θ.

Factorize the joint distribution in the reverse direction, and make each step a neural-Gaussian model:

$$p_\theta(x_{0:T}) = p(x_T) \prod_{t=1}^{T} p_\theta(x_{t-1} \mid x_t), \qquad p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(\mu_\theta(x_t, t),\ \sigma_t^2 \cdot I).$$

We set $p(x_T) = \mathcal{N}(0, I)$. Here σ_t is fixed and we have a closed-form.

Recall that we use neural-Gaussian model in VAE. Our idea is similar here. We set $\sigma_t^2 = \sigma_q^2(t)$ in p_θ.

Central question: what should μ_θ be?

Intractability of the True Reverse Kernel

What is the true conditional distribution derived from forward process? Bayes' rule gives

$$q(x_{t-1} \mid x_t) = \frac{q(x_t \mid x_{t-1}) \cdot q(x_{t-1})}{q(x_t)}.$$

The first factor is the forward kernel: known. But

$$q(x_{t-1}) = \int q(x_{t-1} \mid x_0) \cdot p_{\text{data}}(x_0)\, dx_0$$

is the noisy-image marginal, and it contains p_{data} — the very object we do not know.

Intractability of the True Reverse Kernel

What is the true conditional distribution derived from forward process? Bayes' rule gives

$$q(x_{t-1} \mid x_t) = \frac{q(x_t \mid x_{t-1}) \cdot q(x_{t-1})}{q(x_t)}.$$

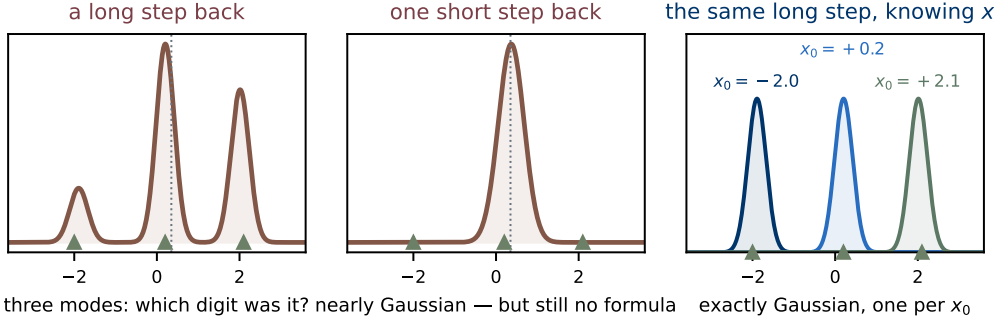
The first factor is the forward kernel: known. But

$$q(x_{t-1}) = \int q(x_{t-1} \mid x_0) \cdot p_{\text{data}}(x_0)\, dx_0$$

is the noisy-image marginal, and it contains p_{data} — the very object we do not know.

So $q(x_{t-1} \mid x_t)$ has no formula, and in general is not even Gaussian.

The Exact Reverse Kernel for a Three-Point Data Set



Exact densities for a data set with three possible values (green triangles). Left: step a long way back from a noisy x_t and the answer is genuinely ambiguous — three candidates. Middle: step a short way and it is nearly Gaussian, but still has no closed form. Right: the same long step, but told which x_0 we came from — now exactly Gaussian.

Three-atom data distribution, $\overline{\alpha}_t = 0.30$, no simulation.

Conditioning on the Clean Sample

Apply Bayes' rule again, but this time hold x_0 fixed:

$$q(x_{t-1} \mid x_t, x_0) = \frac{q(x_t \mid x_{t-1}, x_0) \cdot q(x_{t-1} \mid x_0)}{q(x_t \mid x_0)}$$
$$= \frac{q(x_t \mid x_{t-1}) \cdot q(x_{t-1} \mid x_0)}{q(x_t \mid x_0)}.$$

Conditioning on the Clean Sample

Apply Bayes' rule again, but this time hold x_0 fixed:

$$q(x_{t-1} \mid x_t, x_0) = \frac{q(x_t \mid x_{t-1}, x_0) \cdot q(x_{t-1} \mid x_0)}{q(x_t \mid x_0)}$$
$$= \frac{q(x_t \mid x_{t-1}) \cdot q(x_{t-1} \mid x_0)}{q(x_t \mid x_0)}.$$

The second line drops x_0 by the Markov property: given x_{t-1}, the next state does not look back. Every factor is now a Gaussian distribution.

Conditioning on the Clean Sample

Apply Bayes' rule again, but this time hold x_0 fixed:

$$q(x_{t-1} \mid x_t, x_0) = \frac{q(x_t \mid x_{t-1}, x_0) \cdot q(x_{t-1} \mid x_0)}{q(x_t \mid x_0)}$$
$$= \frac{q(x_t \mid x_{t-1}) \cdot q(x_{t-1} \mid x_0)}{q(x_t \mid x_0)}.$$

The second line drops x_0 by the Markov property: given x_{t-1}, the next state does not look back. Every factor is now a Gaussian distribution.

Key observation: $q(x_{t-1} \mid x_t, x_0)$ **is exactly Gaussian. We show is closed form in next slide; the derivation is in Appendix B.1.**

The Forward-Process Posterior

Under the forward process q, we can write $q(x_{t-1} \mid x_t, x_0)$ in closed-form:

$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}(\mu_q(x_t, x_0),\ \sigma_q^2(t) \cdot I)$$

with

$$\mu_q(x_t, x_0) = \frac{(1-\overline{\alpha}_{t-1})\sqrt{\alpha_t}}{1-\overline{\alpha}_t} \cdot x_t + \frac{(1-\alpha_t)\sqrt{\overline{\alpha}_{t-1}}}{1-\overline{\alpha}_t} \cdot x_0,$$

$$\sigma_q^2(t) = \frac{(1-\alpha_t)(1-\overline{\alpha}_{t-1})}{1-\overline{\alpha}_t}.$$

The mean is a weighted average of x_t and x_0; the variance does not depend on the data at all. We set $\sigma_t^2 = \sigma_q^2(t)$ in p_θ.

The Forward-Process Posterior

Under the forward process q, we can write $q(x_{t-1} \mid x_t, x_0)$ in closed-form:

$$q(x_{t-1} \mid x_t, x_0) = \mathcal{N}(\mu_q(x_t, x_0),\ \sigma_q^2(t) \cdot I)$$

with

$$\mu_q(x_t, x_0) = \frac{(1-\overline{\alpha}_{t-1})\sqrt{\alpha_t}}{1-\overline{\alpha}_t} \cdot x_t + \frac{(1-\alpha_t)\sqrt{\overline{\alpha}_{t-1}}}{1-\overline{\alpha}_t} \cdot x_0,$$

$$\sigma_q^2(t) = \frac{(1-\alpha_t)(1-\overline{\alpha}_{t-1})}{1-\overline{\alpha}_t}.$$

The mean is a weighted average of x_t and x_0; the variance does not depend on the data at all. We set $\sigma_t^2 = \sigma_q^2(t)$ in p_θ.

Now we can go back to derive the loss function for p_θ.

ELBO Derivation

derive

Treat the whole noisy chain $x_{1:T}$ as the latent variable. The likelihood is a marginal, so multiply and divide by the forward process:

$$\log p_\theta(x_0) = \log \int p_\theta(x_{0:T})\, dx_{1:T} = \log\ \mathbb{E}_{q(x_{1:T}|x_0)}\left[\frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)}\right].$$

ELBO Derivation

derive

Treat the whole noisy chain $x_{1:T}$ as the latent variable. The likelihood is a marginal, so multiply and divide by the forward process:

$$\log p_\theta(x_0) = \log \int p_\theta(x_{0:T})\, dx_{1:T} = \log\ \mathbb{E}_{q(x_{1:T}|x_0)}\left[\frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)}\right].$$

log is concave, so Jensen's inequality moves it inside the expectation and can only decrease the value:

$$\boxed{\log p_\theta(x_0)\ \geq\ \mathbb{E}_q\left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)}\right]\ =\ \text{ELBO}}$$

ELBO Derivation

derive

Treat the whole noisy chain $x_{1:T}$ as the latent variable. The likelihood is a marginal, so multiply and divide by the forward process:

$$\log p_\theta(x_0) = \log \int p_\theta(x_{0:T})\, dx_{1:T} = \log\ \mathbb{E}_{q(x_{1:T}|x_0)}\left[\frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)}\right].$$

log is concave, so Jensen's inequality moves it inside the expectation and can only decrease the value:

$$\boxed{\log p_\theta(x_0)\ \geq\ \mathbb{E}_q\left[\log \frac{p_\theta(x_{0:T})}{q(x_{1:T} \mid x_0)}\right]\ =\ \text{ELBO}}$$

The gap is exactly $D_{\mathrm{KL}}(q \,\|\, p_\theta(x_{1:T} \mid x_0))$. Lecture 17 fitted q_ϕ to close it; here q is the forward process, fixed in advance.

Diffusion Is a Variational Autoencoder

A diffusion model is a variational autoencoder whose encoder is specified rather than fitted, and whose decoder is divided into T small steps.

	VAE	diffusion
latent	one code $z \in \mathbb{R}^k$, $k \ll d$	the chain $x_1, \dots, x_T$, each the size of x
encoder	learned $q_\phi(z \mid x)$	fixed $q(x_t \mid x_{t-1})$, no parameters
decoder	one pass $p_\theta(x \mid z)$	T small steps $p_\theta(x_{t-1} \mid x_t)$
objective	the ELBO	the same ELBO
prior term	fight $D_{\mathrm{KL}}(q_\phi \| p)$ during training	a check on T and the schedule

Decomposition of the Bound

Substituting both factorisations and collecting terms, the ELBO splits into three pieces:

$$\text{ELBO} = \underbrace{\mathbb{E}\big[\log p_\theta(x_0 \mid x_1)\big]}_{\text{final step, } t=1} - \underbrace{D_{\mathrm{KL}}\big(q(x_T \mid x_0) \,\|\, p(x_T)\big)}_{\text{no } \theta}$$

$$- \sum_{t=2}^{T} \underbrace{\mathbb{E}\Big[D_{\mathrm{KL}}\big(q(x_{t-1} \mid x_t, x_0) \,\|\, p_\theta(x_{t-1} \mid x_t)\big)\Big]}_{\text{one term per level}}$$

Source: Collecting the terms is bookkeeping; it is written out in Appendix B.2.

Reading the Three Terms

- The sum is the training signal: at each level, make the model's step agree with the posterior step.
- The first term is the same thing at $t = 1$ — the last denoising step, which lands on the clean image. It carries θ too, and the simplified loss will treat it in exactly the same form as the rest.
- The middle term carries no θ, so it cannot be trained. It asks whether the forward process really reaches $\mathcal{N}(0, I)$ by step T — a check on T and the schedule, not on the network.

Reading the Three Terms

- The sum is the training signal: at each level, make the model's step agree with the posterior step.
- The first term is the same thing at $t = 1$ — the last denoising step, which lands on the clean image. It carries θ too, and the simplified loss will treat it in exactly the same form as the rest.
- The middle term carries no θ, so it cannot be trained. It asks whether the forward process really reaches $\mathcal{N}(0, I)$ by step T — a check on T and the schedule, not on the network.

So maximising the bound means, at every level, matching one Gaussian to another. That is the next slide.

Reduction of the KL Divergence to Mean Matching

Give the model the same variance as the posterior found above, $p_\theta(x_{t-1} \mid x_t) = \mathcal{N}(\mu_\theta(x_t, t), \sigma_q^2(t) I)$. For two Gaussians with equal covariance,

$$D_{\mathrm{KL}}(\mathcal{N}(\mu_q, \sigma^2 I) \,\|\, \mathcal{N}(\mu_\theta, \sigma^2 I)) = \frac{1}{2\sigma^2} \|\mu_q - \mu_\theta\|^2$$

So the ELBO term at level t asks us to match the posterior mean. Nothing about divergences survives into the implementation — but one difficulty does, and it is the subject of the next slide.

The Objective Is a Regression

Matching the means means minimising, at each level,

$$\min_{\mu_\theta} \ \mathbb{E}_{x_0, x_t} \| \mu_\theta(x_t) - \mu_q(x_t, x_0) \|^2.$$

The Objective Is a Regression

Matching the means means minimising, at each level,

$$\min_{\mu_\theta} \ \mathbb{E}_{x_0, x_t} \|\mu_\theta(x_t) - \mu_q(x_t, x_0)\|^2.$$

A regression: fit the target $\mu_q(x_t, x_0)$ on the input x_t. Both sides are computable during training, where x_0 is the image we started from.

The Objective Is a Regression

Matching the means means minimising, at each level,

$$\min_{\mu_\theta} \; \mathbb{E}_{x_0, x_t} \|\mu_\theta(x_t) - \mu_q(x_t, x_0)\|^2.$$

A regression: fit the target $\mu_q(x_t, x_0)$ on the input x_t. Both sides are computable during training, where x_0 is the image we started from.

Nothing so far says how to parameterize μ_θ. The next slide shows that most of the target is already known, so the network should only be asked for the rest.

Most of the Target Is Already Known

derive

Use $x_t = \sqrt{\bar{\alpha}_t} \cdot x_0 + \sqrt{1 - \bar{\alpha}_t} \cdot \varepsilon$ to write x_0 through ε. Substituting into μ_q,

$$\mu_q = \underbrace{\frac{1}{\sqrt{\alpha_t}} x_t}_{\text{known from } x_t} - \underbrace{\frac{1}{\sqrt{\alpha_t}} \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \cdot \varepsilon}_{\text{the only unknown}} .$$

Most of the Target Is Already Known

derive

Use $x_t = \sqrt{\bar{\alpha}_t} \cdot x_0 + \sqrt{1-\bar{\alpha}_t} \cdot \varepsilon$ to write x_0 through ε. Substituting into μ_q,

$$\mu_q = \underbrace{\frac{1}{\sqrt{\alpha_t}} x_t}_{\text{known from } x_t} - \underbrace{\frac{1}{\sqrt{\alpha_t}} \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \cdot \varepsilon}_{\text{the only unknown}}.$$

So give μ_θ the same shape — keep the known part, and let a network supply the unknown one:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \cdot \epsilon_\theta(x_t, t) \right).$$

Most of the Target Is Already Known

derive

Use $x_t = \sqrt{\bar{\alpha}_t} \cdot x_0 + \sqrt{1-\bar{\alpha}_t} \cdot \varepsilon$ to write x_0 through ε. Substituting into μ_q,

$$\mu_q = \underbrace{\frac{1}{\sqrt{\alpha_t}} x_t}_{\text{known from } x_t} - \underbrace{\frac{1}{\sqrt{\alpha_t}} \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \cdot \varepsilon}_{\text{the only unknown}}.$$

So give μ_θ the same shape — keep the known part, and let a network supply the unknown one:

$$\mu_\theta(x_t, t) = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1-\alpha_t}{\sqrt{1-\bar{\alpha}_t}} \cdot \epsilon_\theta(x_t, t) \right).$$

The known parts cancel in the difference, and the regression collapses to predicting the noise: $\|\mu_q - \mu_\theta\|^2 \propto \|\varepsilon - \epsilon_\theta(x_t, t)\|^2$.

The Weighted and Simplified Losses

Collecting the $1/(2\sigma_q^2)$ from the KL gives the exact bound,

$$\mathcal{L}(\theta) = \sum_t \mathbb{E}\left[\frac{(1-\alpha_t)^2}{2\sigma_q^2(t)(1-\overline{\alpha}_t)\alpha_t}\|\varepsilon - \epsilon_\theta(x_t, t)\|^2\right].$$

The Weighted and Simplified Losses

Collecting the $1/(2\sigma_q^2)$ from the KL gives the exact bound,

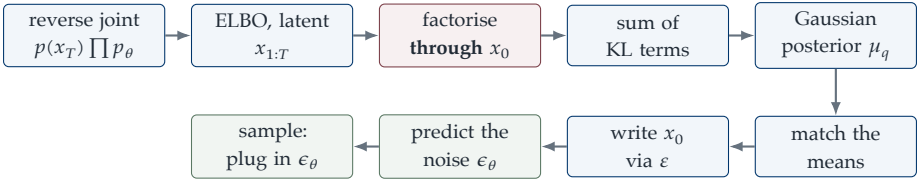
$$\mathcal{L}(\theta) = \sum_t \mathbb{E}\left[\frac{(1-\alpha_t)^2}{2\sigma_q^2(t)(1-\overline{\alpha}_t)\alpha_t}\|\varepsilon - \epsilon_\theta(x_t, t)\|^2\right].$$

Ho et al. found that dropping the weight gives better samples:

$$\boxed{\mathcal{L}_{\text{simple}}(\theta) = \mathbb{E}_{t,x_0,\varepsilon}\left[\|\varepsilon - \epsilon_\theta(\sqrt{\overline{\alpha}_t}\cdot x_0 + \sqrt{1-\overline{\alpha}_t}\cdot\varepsilon,\ t)\|^2\right]}$$

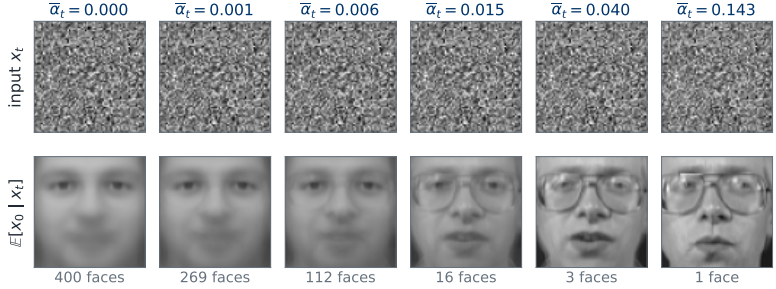
Uniform weights emphasise the high-noise levels relative to the likelihood bound. The model is then better at global structure and slightly worse at likelihood — a deliberate trade, not an oversight.

The Route, in One Picture



Every arrow was an identity or an exact Gaussian calculation. The red box is the only step that is not mechanical, and it is the one worth remembering: keep x_0 in the conditioning so that every factor stays Gaussian. The two green boxes are what survives into code.

What the Loss Is Actually Asking For



The loss just derived is a regression, so its minimiser is $\mathbb{E}[x_0 \mid x_t]$ — and on a finite training set that has a closed form: a softmax-weighted average of the training images. Each label counts how many images the average effectively uses. High noise gives the average face; as the noise falls it collapses onto one training image. The exact optimum memorises; a smooth network generates because it cannot represent it.

Exact minimum-MSE denoiser for the 400 Olivetti faces, cosine schedule.

Outline

The DDPM Training Algorithm

DDPM training: repeat until the loss stops falling

1. Draw a training image x_0;
2. Draw a level $t \sim \mathrm{Unif}\{1, \dots, T\}$ and noise $\varepsilon \sim \mathcal{N}(0, I)$;
3. Form $x_t = \sqrt{\overline{\alpha}_t} \cdot x_0 + \sqrt{1 - \overline{\alpha}_t} \cdot \varepsilon$;
4. Step on $\nabla_\theta \| \epsilon_\theta(x_t, t) - \varepsilon \|^2$.

No chain is simulated, no likelihood is evaluated, and no divergence is computed. After all that variational algebra, the implementation is ordinary supervised regression on pairs we manufacture.

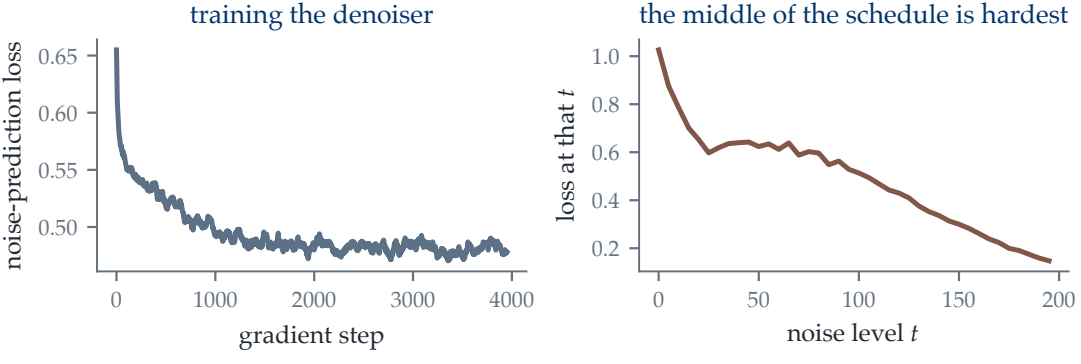
Source: Ho, Jain, and Abbeel (2020), *Denoising Diffusion Probabilistic Models*, Algorithms 1 and 2.

The Training Loop in PyTorch

```
alpha_bar = torch.cumprod(1.0 - betas, dim=0)      # shape (T,)

for x0 in loader:                                  # (N, C, H, W)
    t = torch.randint(0, T, (x0.shape[0],))        # one level per image
    eps = torch.randn_like(x0)

    a = alpha_bar[t].view(-1, 1, 1, 1)             # broadcast over pixels
    xt = a.sqrt() * x0 + (1 - a).sqrt() * eps      # the closed form

    loss = F.mse_loss(model(xt, t), eps)
    opt.zero_grad(); loss.backward(); opt.step()
```

Every image in the batch gets its own noise level, so one pass covers the whole path. Nothing here is sequential.

Loss by Noise Level



Left: the loss over 4,000 Adam steps. Right: the same loss split by level. Both ends are easy — at small t almost nothing was added, at large t the input essentially is the answer — and the middle is where the model earns its keep.

Two-moons target, four-layer network, $T = 200$, batch 512.

The DDPM Sampling Algorithm

DDPM ancestral sampling

1. $x_T \sim \mathcal{N}(0, I)$;
2. For $t = T, \dots, 1$, call the network once and take

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right) + \sigma_t \cdot z, \quad z \sim \mathcal{N}(0, I);$$

3. Return x_0, with $z = 0$ on the final step.

The DDPM Sampling Algorithm

DDPM ancestral sampling

1. $x_T \sim \mathcal{N}(0, I)$;
2. For $t = T, \dots, 1$, call the network once and take

$$x_{t-1} = \frac{1}{\sqrt{\alpha_t}} \left(x_t - \frac{1 - \alpha_t}{\sqrt{1 - \bar{\alpha}_t}} \epsilon_\theta(x_t, t) \right) + \sigma_t \cdot z, \quad z \sim \mathcal{N}(0, I);$$

3. Return x_0, with $z = 0$ on the final step.

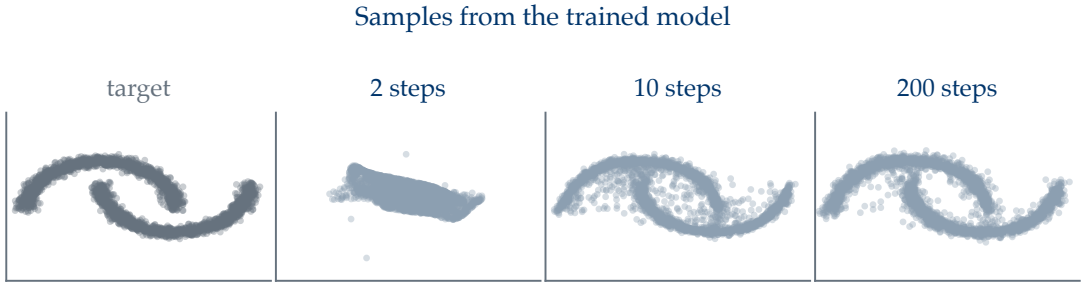
Training touched each image once; sampling calls the network once per step, strictly sequentially. That asymmetry is the practical cost of diffusion, and the subject of the next lecture.

The Sampling Loop in PyTorch

```
x = torch.randn(n, C, H, W)                          # start in pure noise

for t in reversed(range(T)):
    eps = model(x, torch.full((n,), t))              # one call per step
    a, a_bar = alphas[t], alpha_bar[t]

    mean = (x - (1 - a) / (1 - a_bar).sqrt() * eps) / a.sqrt()
    if t > 0:
        var = (1 - a) * (1 - alpha_bar[t - 1]) / (1 - a_bar)
        x = mean + var.sqrt() * torch.randn_like(x)
    else:
        x = mean                                     # no noise on the last step
```

The loop body is four lines of arithmetic around one network call. All of the modelling is in `model`; all of the cost is in the loop.

Samples from the Trained Model



Two steps collapse the cloud into a blur. Ten steps recover both moons with a haze between them. Two hundred steps are visually indistinguishable from the target — so the step count is a choice with a price, not a constant of the method.

3,000 samples per panel from the same trained network.

Outline

What the Denoising Network Has to Do

The loss is $\|\varepsilon - \epsilon_\theta(x_t, t)\|^2$, so the network's signature is fixed by the problem:

$$\epsilon_\theta : \underbrace{\mathbb{R}^{C\times H\times W}}_{\text{noisy image}} \times \underbrace{\{1, \dots, T\}}_{\text{noise level}} \longrightarrow \underbrace{\mathbb{R}^{C\times H\times W}}_{\text{predicted noise}} .$$

What the Denoising Network Has to Do

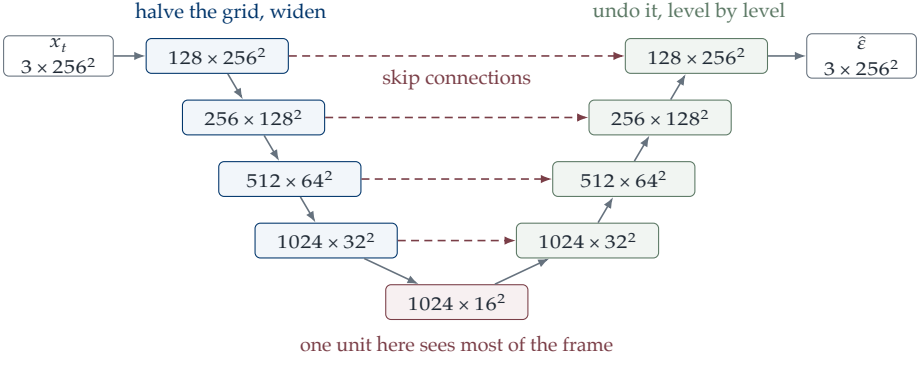
The loss is $\|\varepsilon - \epsilon_\theta(x_t, t)\|^2$, so the network's signature is fixed by the problem:

$$\epsilon_\theta : \underbrace{\mathbb{R}^{C\times H\times W}}_{\text{noisy image}} \times \underbrace{\{1, \dots, T\}}_{\text{noise level}} \longrightarrow \underbrace{\mathbb{R}^{C\times H\times W}}_{\text{predicted noise}} .$$

Three requirements follow, and they pull in different directions:

1. **Same shape.** The output is an image, pixel-aligned with the input — not a vector, not a class.
2. **Local detail and global structure.** The noise at one pixel is guessed from nearby pixels and from what the whole frame shows — is this a face or a landscape?
3. **The level t.** One set of weights must serve every noise level, so t has to enter as an input.

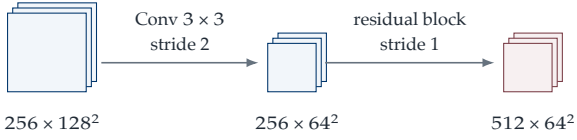
One Example Configuration



Source: Boxes read channels × grid. **One configuration among many** — depths, widths and resolutions all vary between implementations. Shape after Ronneberger et al. (2015); widths and convolutions after Ho et al. (2020).

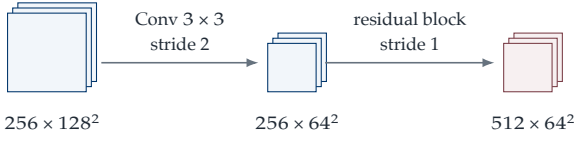
The Encoder

At each level: convolutions at a fixed resolution, then one stride-2 convolution that halves the grid.



The Encoder

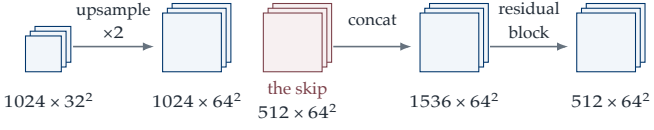
At each level: convolutions at a fixed resolution, then one stride-2 convolution that halves the grid.



The grid shrinks and the channels grow, for two reasons: (a) a unit at low resolution sees much more of the image, so there is more to describe; and (b) since a convolution costs about $H \cdot W \cdot C_{\text{in}} \cdot C_{\text{out}} \cdot k^2$, quartering $H \cdot W$ while doubling the channels leaves the cost per level roughly unchanged.

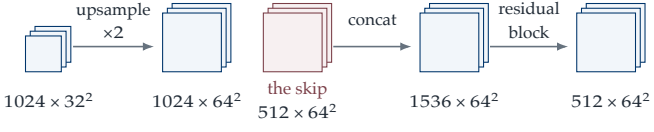
The Decoder

Its reverse, one level at a time: upsample, bring in the matching encoder output, then reduce the channels again.



The Decoder

Its reverse, one level at a time: upsample, bring in the matching encoder output, then reduce the channels again.



Concatenating adds channels, $1024 + 512 = 1536$, and the block that follows brings them back down. That is where the skip connection enters, returning the detail the encoder threw away.

Source: Upsampling is classically a transposed convolution; Appendix C.

Two Kinds of Residual Block

Every level is built from the same unit: two convolutions, with the input added back at the end. The only question is what to do when the channel counts do not match.

Channels unchanged

$h + F(h)$ — the shortcut is the identity. Most blocks are of this kind

Channels change

$Wh + F(h)$, where W is a 1×1 convolution that matches the counts. Used at the start of a level, and after a concatenation

Two Kinds of Residual Block

Every level is built from the same unit: two convolutions, with the input added back at the end. The only question is what to do when the channel counts do not match.

Channels unchanged

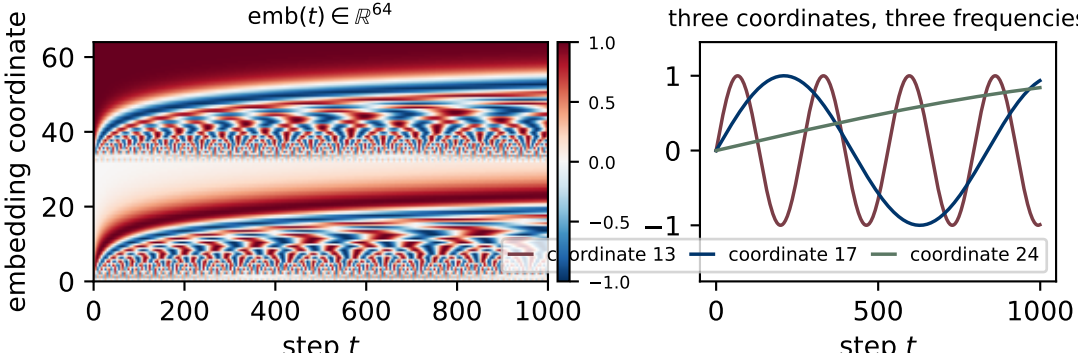
$h + F(h)$ — the shortcut is the identity. Most blocks are of this kind

Channels change

$Wh + F(h)$, where W is a 1×1 convolution that matches the counts. Used at the start of a level, and after a concatenation

This is the ResNet shortcut of Lecture 13, unchanged. The noise level t also enters inside these blocks — the next two slides.

The Sinusoidal Time Embedding



The integer t is expanded into sinusoids at many frequencies. Fast coordinates separate adjacent levels; slow coordinates say roughly where on the path we are. A raw scalar t/T would give the network one nearly constant input to work with.

The standard 64-dimensional sinusoidal embedding.

How the Noise Level Enters

Once per forward pass, the integer t becomes a vector:

$$t \longmapsto \text{emb}(t) \in \mathbb{R}^{128} \longmapsto \text{2-layer MLP} \longmapsto e_t \in \mathbb{R}^{512}.$$

How the Noise Level Enters

Once per forward pass, the integer t becomes a vector:

$$t \longmapsto \text{emb}(t) \in \mathbb{R}^{128} \longmapsto \text{2-layer MLP} \longmapsto e_t \in \mathbb{R}^{512}.$$

Then inside every block, a small linear layer of its own maps e_t to one number per output channel, which is added to the feature map:

$$h \longleftarrow h + W_\ell\, e_t, \qquad W_\ell\, e_t \in \mathbb{R}^{C_{\text{out}}},$$

broadcast over H and W — the same offset at every pixel.

How the Noise Level Enters

Once per forward pass, the integer t becomes a vector:

$$t \longmapsto \text{emb}(t) \in \mathbb{R}^{128} \longmapsto \text{2-layer MLP} \longmapsto e_t \in \mathbb{R}^{512}.$$

Then inside every block, a small linear layer of its own maps e_t to one number per output channel, which is added to the feature map:

$$h \longleftarrow h + W_\ell\, e_t, \qquad W_\ell\, e_t \in \mathbb{R}^{C_{\text{out}}},$$

broadcast over H and W — the same offset at every pixel.

So t never changes the weights; it shifts the channels — one network therefore serves all T levels.

Source: Ho et al. add a shift; most implementations since emit *two* vectors and apply $h \leftarrow (1 + \text{scale}) \cdot h + \text{shift}$.

The Residual Block in PyTorch

```
class ResBlock(nn.Module):
    def __init__(self, c_in, c_out, t_dim):
        super().__init__()
        self.conv1 = nn.Conv2d(c_in,  c_out, 3, padding=1)
        self.conv2 = nn.Conv2d(c_out, c_out, 3, padding=1)
        self.time  = nn.Linear(t_dim, c_out)      # t -> per-channel shift
        self.skip  = nn.Conv2d(c_in, c_out, 1)    # match shapes for the residual

    def forward(self, x, t_emb):
        h = self.conv1(F.silu(self.norm1(x)))
        h = h + self.time(F.silu(t_emb))[:, :, None, None]
        h = self.conv2(F.silu(self.norm2(h)))
        return h + self.skip(x)
```

The U-Net is this block repeated, with `torch.cat` at each skip.

Summary: What Diffusion Does

$$q(x_t \mid x_0) = \mathcal{N}(\sqrt{\overline{\alpha}_t} \cdot x_0,\ (1 - \overline{\alpha}_t)I), \qquad \min_\theta\ \mathbb{E}_{t, x_0, \varepsilon} \|\epsilon_\theta(x_t, t) - \varepsilon\|^2.$$

1. A path from data to noise. Going forward is arithmetic; only coming back is learned.
2. The true reverse step has no formula — but conditioning on x_0 makes it Gaussian, and the ELBO reduces to matching its mean.
3. Most of that mean is known from x_t, so all that is left is predicting the noise: one line of regression.

Summary: Diffusion against the VAE

Both turn noise into an image; they differ in how much one network call has to do.

VAE

one call, $z \mapsto g_\theta(z)$. Squared error makes the output $\mathbb{E}[x \mid z]$: an average over all z leaves undetermined

Diffusion

T calls, each predicting the noise in an almost-clean image, so the averaging is over a small residual and detail survives

Summary: Diffusion against the VAE

Both turn noise into an image; they differ in how much one network call has to do.

VAE

one call, $z \mapsto g_\theta(z)$. Squared error makes the output $\mathbb{E}[x \mid z]$: an average over all z leaves undetermined

Diffusion

T calls, each predicting the noise in an almost-clean image, so the averaging is over a small residual and detail survives

The idea is decomposition: one hard generation problem becomes many easy denoising problems, at the price of T network calls.

Source: Opening digits: edge strength 0.957 real, 0.786 VAE, 0.901 diffusion; reproduced in the demo notebook.

Next Lecture

We can now generate images — slowly, at low resolution, and with no way to say what we want.

Lecture 19 — Latent Diffusion, Flow Matching, and Text fixes all three: DDIM cuts the step count by an order of magnitude, a learned latent space makes each step cheap, text conditioning says what to draw, and flow matching replaces the noising schedule with a straight line.

Reading for this lecture: Ho, Jain, and Abbeel (2020), *Denoising Diffusion Probabilistic Models*.

Appendix

Five groups of notes skipped in the lecture hour.

A. **Notation and the forward process.**

B. **The bound and the loss.** The posterior, the ELBO, and mean matching as the noise loss.

C. **Upsampling with a convolution.** The transposed convolution, and why diffusion interpolates.

D. **Another reading.** The noise prediction as a score.

E. **Further reading.**

A.1 Notation Used in Both Lectures

symbol	meaning	who chooses it
x_0	a clean data point	the data
x_t	the state at noise level t	the forward process
T	number of levels, often 1000	us
β_t	noise added at step t	us (the schedule)
$\alpha_t = 1 - \beta_t$	signal kept in one step	follows
$\overline{\alpha}_t = \prod_{s \le t} \alpha_s$	signal kept in total	follows
q	the forward (noising) law	fixed, known
p_θ	the reverse (denoising) law	learned
$\epsilon_\theta(x_t, t)$	the network	learned

Everything written with q is arithmetic. Everything written with θ is a fit.

A.2 Proof of the Closed Form

derive

Assume the claim at $t-1$, so $x_{t-1} = \sqrt{\overline{\alpha}_{t-1}} \cdot x_0 + \sqrt{1-\overline{\alpha}_{t-1}}\,\tilde{\varepsilon}$. Substituting into one forward step,

$$x_t = \underbrace{\sqrt{\alpha_t \overline{\alpha}_{t-1}}}_{=\sqrt{\overline{\alpha}_t}} \cdot x_0 + \underbrace{\sqrt{\alpha_t(1-\overline{\alpha}_{t-1})}\,\tilde{\varepsilon} + \sqrt{1-\alpha_t} \cdot \varepsilon_t}_{\text{two independent Gaussians}}.$$

Independent Gaussians add in variance, so the noise term is Gaussian with variance

$$\alpha_t(1-\overline{\alpha}_{t-1}) + (1-\alpha_t) = 1 - \alpha_t\overline{\alpha}_{t-1} = 1 - \overline{\alpha}_t.$$

This identity is what makes the supervision free: every training pair (x_t, ε) follows from a clean image in closed form.

A.3 The Two Noise Schedules

derive

Linear (Ho et al., 2020) fixes the per-step noise directly,

$$\beta_t = \beta_1 + \frac{t-1}{T-1} \cdot (\beta_T - \beta_1), \qquad \beta_1 = 10^{-4},\ \beta_T = .02,\ T = 1000,$$

and the signal follows as $\bar{\alpha}_t = \prod_{s \le t}(1 - \beta_s)$.

Cosine (Nichol and Dhariwal, 2021) fixes the signal instead,

$$\bar{\alpha}_t = \frac{f(t)}{f(0)}, \qquad f(t) = \cos^2\Big(\frac{\pi}{2} \cdot \frac{t/T + s}{1+s}\Big), \qquad s = .008,$$

and recovers the per-step noise as $\beta_t = 1 - \bar{\alpha}_t / \bar{\alpha}_{t-1}$, clipped at .999.

A.4 Why It Is Called "Cosine"

The name describes the signal curve $\bar{\alpha}_t$, not the noise curve β_t. In

$$f(t) = \cos^2\left(\frac{\pi}{2} \cdot \frac{t/T + s}{1 + s}\right),$$

the argument sweeps from 0 to $\pi/2$ as t runs from 0 to T: one quarter period of a cosine, squared, so f falls from 1 to 0.

The offset $s = .008$ shifts the start slightly off the flat top, which keeps β_1 from being nearly zero.

The shape of β_t is then whatever is left over — which is why it runs away near $t = T$ and has to be clipped at .999.

A.5 Why the Cosine Was Proposed

derive

Take logs of the linear schedule. Since β_t is small,

$$\log \bar{\alpha}_t = \sum_{s \le t} \log(1 - \beta_s) \approx -\sum_{s \le t} \beta_s,$$

and the partial sum of a linear ramp grows like t^2. So $\bar{\alpha}_t$ decays like e^{-ct^2}: super-exponentially.

By $t \approx 700$ the signal is numerically gone, so the last 300 steps train the network on inputs that are already pure noise.

The cosine is chosen so that $\bar{\alpha}_t$ falls almost linearly through the middle of its range, spreading the difficulty across all T steps. The price is that β_t must grow sharply near $t = T$, which is why it is clipped.

B.1 Derivation of the Forward-Process Posterior

Both factors are Gaussian in x_{t-1}:

$$q(x_{t-1} \mid x_t, x_0) \;\propto\; \underbrace{q(x_t \mid x_{t-1})}_{\mathcal{N}(\sqrt{\alpha_t}\cdot x_{t-1},\,(1-\alpha_t)I)} \;\cdot\; \underbrace{q(x_{t-1} \mid x_0)}_{\mathcal{N}(\sqrt{\overline{\alpha}_{t-1}}\cdot x_0,\,(1-\overline{\alpha}_{t-1})I)} \;.$$

Adding the two quadratic forms, the precision is

$$\frac{1}{\sigma_q^2} = \frac{\alpha_t}{1-\alpha_t} + \frac{1}{1-\overline{\alpha}_{t-1}} = \frac{1-\overline{\alpha}_t}{(1-\alpha_t)(1-\overline{\alpha}_{t-1})},$$

using $\overline{\alpha}_t = \alpha_t \overline{\alpha}_{t-1}$. The mean is the precision-weighted average of the two centres, which is μ_q.

B.2 The ELBO Decomposition, Term by Term

Writing $\log \frac{p_\theta(x_{0:T})}{q(x_{1:T}|x_0)}$ with the x_0-conditioned factorisation and separating,

$$\begin{aligned}\text{ELBO} = \mathbb{E}_q[\log p_\theta(x_0 \mid x_1)] + \mathbb{E}_q\left[\log \frac{p(x_T)}{q(x_T \mid x_0)}\right] \\ + \sum_{t=2}^{T} \mathbb{E}_q\left[\log \frac{p_\theta(x_{t-1} \mid x_t)}{q(x_{t-1} \mid x_t, x_0)}\right].\end{aligned}$$

The second term is $-D_{\mathrm{KL}}(q(x_T \mid x_0)\|p(x_T))$ and carries no θ. Each summand in the third is $-\mathbb{E}_{q(x_t|x_0)} D_{\mathrm{KL}}(q(x_{t-1} \mid x_t, x_0)\|p_\theta(x_{t-1} \mid x_t))$.

B.3 From Mean Matching to the Noise Loss

With the shared variance $\sigma_q^2(t)$, both means share the term $x_t/\sqrt{\alpha_t}$, so

$$\mu_q - \mu_\theta = -\frac{1-\alpha_t}{\sqrt{\alpha_t}\cdot\sqrt{1-\overline{\alpha}_t}}(\varepsilon - \epsilon_\theta(x_t, t)),$$

and therefore

$$D_{\mathrm{KL}} = \frac{(1-\alpha_t)^2}{2\sigma_q^2(t)\cdot\alpha_t(1-\overline{\alpha}_t)}\|\varepsilon - \epsilon_\theta(x_t, t)\|^2.$$

Setting the bracket to one gives $\mathcal{L}_{\text{simple}}$.

B.4 The Clean-Image Estimate

The network predicts ε, but the implied clean image follows from the same forward identity:

$$\hat{x}_0(x_t, t) = \frac{x_t - \sqrt{1 - \overline{\alpha}_t} \cdot \epsilon_\theta(x_t, t)}{\sqrt{\overline{\alpha}_t}}.$$

When $\overline{\alpha}_t$ is small the division amplifies any error in ϵ_θ, which is why $\hat{x}_0$ is usually clamped to the valid intensity range before it is used. Lecture 19 builds its fast sampler directly on this quantity.

C.1 Upsampling with a Convolution

Convolution shrinks a grid: stride s turns n input positions into about n/s outputs. Going the other way needs an operation that turns n positions into sn.

C.1 Upsampling with a Convolution

Convolution shrinks a grid: stride s turns n input positions into about n/s outputs. Going the other way needs an operation that turns n positions into sn.

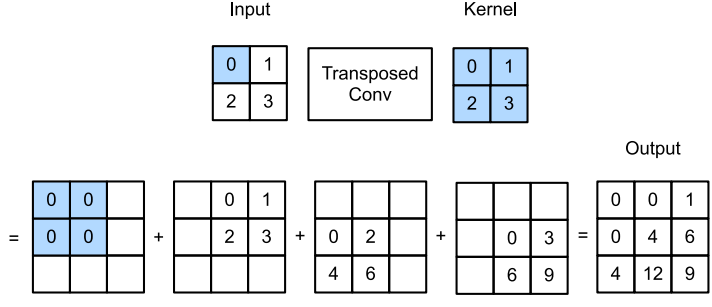
The **transposed convolution** reverses the roles. Rather than sliding a window and writing one output, it takes each input value, scales the whole kernel by it, and adds that copy into the output:

$$Y[i : i + k,\ j : j + k] \ \mathrel{+}= \ X[i, j] \cdot K.$$

With stride 1 and no padding, an $n \times n$ input and a $k \times k$ kernel give an $(n + k - 1) \times (n + k - 1)$ output. Stride s and padding p give $s(n - 1) - 2p + k$.

Source: Following *Dive into Deep Learning*, section 14.10. With $k = 3$, $s = 2$ the copies overlap unevenly and print as a checkerboard, so diffusion models repeat each value over a 2×2 block and convolve instead (Odena et al., 2016).

C.2 A Worked Transposed Convolution



Each input value scatters a scaled copy of the kernel; summing the four intermediate tensors gives the output.

Source: Figure from Zhang, Lipton, Li and Smola, *Dive into Deep Learning*, section 14.10, CC BY-SA 4.0.

C.3 Why It Is Called “Transposed”

derive

Flatten the image. An ordinary convolution is a linear map, so it can be written as a sparse matrix acting on the flattened input:

$$y = Wx, \qquad W \in \mathbb{R}^{m \times n}, \quad m < n.$$

C.3 Why It Is Called “Transposed”

derive

Flatten the image. An ordinary convolution is a linear map, so it can be written as a sparse matrix acting on the flattened input:

$$y = Wx, \qquad W \in \mathbb{R}^{m \times n}, \quad m < n.$$

The transposed convolution is the map with the transposed matrix,

$$\boxed{z = W^{\top} y, \qquad W^{\top} \in \mathbb{R}^{n \times m}}$$

which is why it carries the output shape back to the input shape — and it is exactly what backpropagation already computes through a convolution.

Source: The weights are learned separately; only the sparsity pattern is shared.

D.1 The Score-Function Interpretation

Differentiate the Gaussian $\log q(x_t \mid x_0)$ in x_t:

$$\nabla_{x_t} \log q(x_t \mid x_0) = -\frac{x_t - \sqrt{\bar{\alpha}_t} \cdot x_0}{1 - \bar{\alpha}_t} = \boxed{-\frac{\varepsilon}{\sqrt{1 - \bar{\alpha}_t}}}$$

A network trained to guess the noise is, up to a known factor, estimating the **score** $\nabla_x \log q(x)$ — the direction in which the noisy density increases fastest.

This is the geometric content of the reverse process: moving along the score carries a random point toward regions of higher data density.

E.1 Further Reading

- Ho, Jain, Abbeel (2020), *Denoising Diffusion Probabilistic Models* — the algorithm in this lecture.
- Nichol, Dhariwal (2021), *Improved DDPM* — the cosine schedule and learned variances.
- Luo (2022), *Understanding Diffusion Models: A Unified Perspective* — the ELBO derivation at greater length.
- Nakkiran, Bradley, Zhou, Advani (2024), *Step-by-Step Diffusion* — a derivation avoiding the variational machinery.
- Rogge, Rasul (2022), *The Annotated Diffusion Model* — implemented line by line.
- *Dive into Deep Learning*, section 14.10 — transposed convolution.